Amendments to the Specificati n:

Rewrite the fifth paragraph on page 8 as follows:

B1

Fig. 4a depicts an example of a 32-bit Opcode ~~showing the incorporation of instructions relating to *src1* and *src2*~~ instruction to perform a relative branch with NOPs (BNOP) operation; and **Fig. ~~6b~~ 4b** depicts the pipeline format for performing a relative BNOP operation; and

Rewrite the sixth paragraph on page 8 as follows:

B2

Fig. ~~6a~~ 5a depicts an example of a 32-bit Opcode ~~showing the incorporation of instructions relating to *src2* and *src1*~~ instruction to perform an absolute BNOP operation~~,~~; and **Fig. ~~6b~~ 5b** depicts the pipeline format for performing an absolute BNOP operation.

Rewrite the first paragraph at the top of page 9 as follows:

B3

Data processing devices suitable for use with and incorporating this invention are described in U.S. Patent Application Serial No. ~~_____ (Attorney Docket No. TI-30302), filed February 18, 2000,~~ 09/703,096 entitled "Microprocessor with Improved Instruction Set Architecture"~~, which~~ and is incorporated herein by reference. In an embodiment of the present invention, there are 64 general-purpose registers. General purpose registers A0, A1, A2, B0, B1 and B2 each may be used as a conditional register. Further, each .D unit may load and store double words (64 bits). The .D units may access words and double words on any byte boundary. The .D unit supports data as well as address cross paths. The same register may be used as a data path cross operand for more than one functional unit in an execute packet. A delay clock cycle is introduced when an instruction attempts to read a register via a cross path that was updated in the previous cycle. Up to two long sources and two long results may be accessed on each data path every cycle.

Rewrite the first complete paragraph at page 10 as follows:

In microprocessor 1 there are shown a central processing unit (CPU) 10, data memory 22, program memory 23, peripherals 60 and an external memory interface (EMIF) with a direct memory access (DMA) 61. CPU 10 further has an instruction fetch/decode unit 10a-c, a plurality of execution units, including an arithmetic and load/store unit D1, a multiplier M1, an ALU/shifter unit S1, an arithmetic logic unit ("ALU") L1, a shared multi-port register file 20a from which data are read and to which data are written. Instructions are fetched by fetch unit 10a from instruction memory 23 over a set of busses 41. Decoded instructions are provided from the instruction fetch/decode unit 10a-c to the functional units D1, M1, S1, and L1 over various sets of control lines which are not shown. Data are provided to/from the register file 20a from/to to load/store ~~units~~ unit D1 over a first set of busses 32a, to multiplier M1 over a second set of busses 34a, to ALU/shifter unit S1 over a third set of busses 36a and to ALU L1 over a fourth set of busses 38a. Data are provided to/from the memory 22 from/to the load/store ~~units~~ unit D1 via a fifth set of busses 40a. Note that the entire data path described above is duplicated with register file 20b and execution units D2, M2, S2, and L2. Load/store unit D2 similarly interfaces with memory 22 via a second set of busses. Instructions are fetched by fetch unit 10a from instruction memory 23 over a set of busses 41. Emulation circuitry 50 provides access to the internal operation of integrated circuit 1 which may controlled by an external test/development system (XDS) 51.

B4

Rewrite the last paragraph at the bottom of page 10 and continuing to page 11 as follows:

When microprocessor 1 is incorporated in a data processing system, additional memory or peripherals may be connected to microprocessor 1, as illustrated in Figure 1. For example, Random Access Memory (RAM) 70, a Read Only Memory (ROM) 71 and a Disk 72 are shown connected via an external bus 73. Bus 73 is connected to the External Memory Interface (EMIF) which is part of functional block 61 within microprocessor ~~42~~ 1. A Direct Memory Access (DMA) controller is also included within block 61. The DMA controller part of functional block 61 connects to data memory 22 via a bus and is generally used to move data between memory

B5

B5

and peripherals within microprocessor 1 and memory and peripherals which are external to microprocessor 1.

Rewrite the first complete paragraph at page 11 as follows:

Each functional unit reads directly from and writes directly to the register file within its own data path. That is, the .L1, .S1, .D1, and .M1 units write to register file A 20a and the .L2, .S2, .D2, and .M2 units write to register file B 20b. The register files are connected to the opposite-side register file's functional units via the 1X and 2X cross paths. These cross paths allow functional units from one data path to access a 32-bit operand from the opposite side's register file. The 1X cross path allows data path A's functional units to read their source from register file B. Similarly, the 2X cross path allows data path B's functional units to read their source from register file A.

B6

Insert the following new paragraph after the second complete paragraph at page 11 as follows:

S2 unit may write to control register file 102 from a dst output via a bus (not shown). S2 unit may read from control register file 102 to its *src2* input via a bus (not shown).

B7

Rewrite the second complete paragraph at page 12 as follows:

Fig. 2 is a top level block diagram of ~~a~~ an A unit group ~~78~~, which supports a portion of the arithmetic and logic operations of DSP core ~~44~~ 10. A unit group ~~78~~ handles a variety of operation types requiring a number of functional units including A adder unit 128, A zero detect unit 130, A bit detection unit 132, A R/Z logic unit 134, A pack/replicate unit 136, A shuffle unit 138, A generic logic block unit 140, and A div-seed unit 142. Partitioning of the functional sub-units is based on the functional requirements of A unit group ~~78~~, emphasizing maximum performance while still achieving low power goals. There are two input muxes 144 and 146 for the input operands, both of which allow routing of operands from one of five sources. Both

B8

muxes have three hotpath sources from the A, C and S result busses, and a direct input from register file ~~76~~ RF in the primary datapath. In addition, *src1* mux 144 may pass constant data from decode unit ~~62~~ (not shown), while *src2* mux 146 provides a path for operands from the opposite datapath. Result mux 148 is split into four levels. Simple operations which complete early in the clock cycle are pre-muxed in order to reduce loading on the critical final output mux. A unit group ~~78~~ also is responsible for handling control register operations 143. Although no hardware is required, these operations borrow the read and write ports of A unit group ~~78~~ for routing data. The *src2* read port is used to route data from register file ~~76~~ (RF) to valid configuration registers. Similarly, the write port is borrowed to route configuration register data to register file ~~76~~ RF.

Rewrite the last paragraph at the bottom of page 12 and continuing to page 13 as follows:

Fig. 3 is a top level block diagram of S unit group ~~82~~, which is optimized to handle shifting, rotating, and Boolean operations, although hardware is available for a limited set of add and subtract operations. S unit group ~~82~~ is unique in the most of the hardware may be directly controlled by the programmer. S unit group ~~82~~ has two more read ports than the A and C unit groups, thus permitting instructions to operate on up to four source registers, selected through input muxes 144, 146, 161, and 163. Similar to the A and C unit groups, the primary execution functionality is performed in the Execute cycle of the design. S unit group ~~82~~ has two major functional units: 32-bit S adder unit 156, and S rotate/Boolean unit 165. S rotate/Boolean unit 165 includes S rotator unit 158, S mask generator unit 160, S bit replicate unit 167, S unpack/ sign extend unit 169, and S logical unit 162. The outputs from S rotator unit 158, S mask generator unit 160, S bit replicate unit 167, and S unpack/ sign extend unit 169 are forwarded to S logical unit 162. The various functional units that make up S rotate/Boolean unit 165 may be utilized in combination to make S unit group ~~82~~ capable of handling very complex Boolean operations. Finally, result mux 148 selects an output from one of the two major functional units, S adder unit 156 and S rotate/Boolean unit 165, for forwarding to register file ~~76~~ RF.

Rewrite the first complete paragraph at page 14 as follows:

Although the method and apparatus of this invention may be used with either load or branch instructions, the branch instructions tend to have more room to receive the additional NOP field. Thus, in a method for reducing total code size during branching, the method may comprise the steps of determining a latency in a shift between a first pipelined operation and a second pipelined operation. The latency may be determined by identifying the branch instruction and the first and second pipelined operations. Further the method may conclude by adding a NOP field to an end of the branch instruction, e.g., B label, 5. In determining the latencies within a code, the code ay be manually or automatically searched to locate sections of code, such as branch operations which will necessitate latencies or delays. Alternatively, a particular program may be run and analyzed to determine ~~whether~~ the latencies within the program.

Rewrite the first complete paragraph at page 15 as follows:

The invention will be further clarified by a consideration of the following examples, which are intended to be purely exemplary of the use of the invention. As demonstrated by the following examples, the NOP operation may be encoded into or onto the instruction, such that the NOP is an operation issued in parallel with the instruction requiring the latency. Referring to the examples set forth above, the following examples show the code rewritten according to the present invention:

Example 1b:

```
LD *a0, a5 ,4        % "4" (i.e., four (4) cycles or delay slots) is the NOP field
ADD a5, 6, a7        % a5 value available
```

Example 2b:

```
B label, 5           % "5" (i.e., five (5) cycles or delay slots) is the NOP field
;                    % branch occurs
```

As may be seen from these examples, the NOP field is an instruction operand that ranges from 0 to the maximum latency of the instruction. Nevertheless, other ranges may be applied that may result in further savings on op-code encoding space. Another example is provided below for the

LD instruction of Example 1b, in which a value less than maximum latency is used because other instructions are to be scheduled in the instruction's delay slots.

Example 1c:

```
LD*a0, a5, 3          % "3" (i.e., three (3) cycles or delay slots) is the NOP field
ADD a3, 5, a3         % a new instruction is inserted into the 4th delay slot
ADD ~~a3~~ a5, 6, a7  % a5 value available
```

Rewrite the last paragraph at the bottom of page 15 and continuing to page 16 as follows:

In still another embodiment of the invention, the latency may be identified within a Branch instruction performing a relative branch with NOPs, i.e., a BNOP. An operation code or Opcode may be the first byte of the machine code that describes a particular type of operation and the combination of operands to the central processing unit (CPU). For example, the Opcode for the BNOP instruction may be formed by the combination of a BNOP (*.unit*) code coupled with the identification of a starting source (*src2*) and an ending source (*src1*) code, e.g., *.unit=.S1,.S2*. In this format, the *src2* Opcode map field is used for the *scst12* operand-type unit to perform a relative branch with NOPs using the 12-bit signed constant specified by *src2*. The constant is shifted two (2) bits to the left, then added to the address of the first instruction of the fetch packet that contains the BNOP instruction. Referring to **Fig. 4a**, an example of a 32-bit Opcode is depicted showing the ~~incorporation of instructions relating to *src2* and *src1*~~ BNOP instruction.

Rewrite the last paragraph at the bottom of page 16 and continuing to page 17 as follows:

Only one branch instruction may be executed per cycle. If two (2) branch condition controls are in the same execute packet, i.e., a block of instructions that execute in parallel, and if both are accepted, the program behavior is undefined. Further, when a predicated BNOP instruction is used with a NOP count greater than five (5), a C64X processor, available from Texas Instruments, Inc., of Dallas Texas, will insert the total number of delay slots requested,

only when the predicated condition is false. For example, the following set of instructions insert seven (7) cycles of NOPs into the BNOP instruction:

```
      ZERO .L1    A0
[A0]  BNOP .S1    LABEL,7.
```

B13

Thus, the branch is not taken-, and seven (7) cycles of NOPs are inserted. Conversely, when a predicated BNOP instruction is used with a NOP count greater than five (5) and the predication condition is true, the branch will be taken and the multi-cycle NOP will be simultaneously terminated. ~~For example, .~~—For example, the following set of instructions insert only five (5) cycles of NOPs into the BNOP instruction:

```
      MVK  .D1    1,A0
[A0]  BNOP .S1    LABEL,7.
```

Thus, the branch is taken, and five (5) cycles of NOPs are effectively inserted.

Rewrite the last paragraph at the bottom of page 17 and continuing to page 18 as follows:

In yet another embodiment of this invention, an operation code or Opcode again may be the first byte of the machine code that describes a particular type of operation and the combination of operands to the central processing unit (CPU). For example, the Opcode for the BNOP instruction again may be formed by the combination of a BNOP (*.unit*) code coupled with

B14

the identification of a ~~starting~~ second source (*src2*) and ~~an ending~~ a first source (*src1*) code~~, e.g., .unit=.S2~~. In this format, the *src2* Opcode map field is used for the *xunit* operand-type unit to perform a absolute branch with NOPs. The register specified in *src2* is placed in the program fetch counter (PFC), described above. The 3-bit unsigned constant specified in *src1*, provides the number of delay slots NOPs to be inserted, e.g., from zero (0) to five (5).~~—).~~ Thus, for example, with *src1*=0, no delay slot NOPs are inserted. Consequently, this instruction also reduces the number of instructions required to perform a branch operation when NOPs are required to fill the delay slots of a branch. Referring to **Fig. 5a**, an example of a 32-bit Opcode is depicted showing the incorporation of instructions relating to *src2* and *src1*.

Rewrite the last paragraph at the bottom of page 18 and continuing to page 19 as follows:

As noted above, only one branch instruction may be executed per cycle. If two (2) branch condition controls are in the same execute packet and if both are accepted, the program behavior is undefined. Further, when a predicated BNOP instruction is used with a NOP count greater than five (5), a C64X processor, available from Texas Instruments, Inc., of Dallas Texas, will insert the total number of delay slots requested, only when the predicated condition is false. For example, the following set of instructions insert seven (7) cycles of NOPs into the BNOP instruction:

b15

```
        ZERO .L1    A0
[A0]    BNOP .S1    B3,7.
```

Thus, the branch is not ~~taken~~ taken, and seven (7) cycles of NOPs are inserted. Conversely, when a predicated BNOP instruction is used with a NOP count greater than five (5) and the predication condition is true, the branch will be taken and the multi-cycle NOP will be simultaneously terminated. ~~For example,~~ For example, the following set of instructions insert only five (5) cycles of NOPs into the BNOP instruction:

```
        MVK  .D1    1,A0
[A0]    BNOP .S1    B3,7.
```

Thus, the branch is taken, and five (5) cycles of NOPs are effectively inserted.